ISO 29990:2010

NATIONAL INFOCOMM COMPETENCY FRAMEWORK

NUS National University of Singapore

ISS INSTITUTE OF SYSTEMS SCIENCE

Text Processing using Machine Learning

Deep Learning Foundations

Liling Tan

22 Jan 2019

OVER 5,500 GRADUATE ALUMNI

OFFERING OVER 120 ENTERPRISE IT, INNOVATION & LEADERSHIP PROGRAMMES

TRAINING OVER 120,000 DIGITAL LEADERS & PROFESSIONALS

© 2018 National University of Singapore. All Rights Reserved

Overview

Lecture

- **More Deep Learning Foundations**
  - Recap: Behind the PyTorch Magic (15 mins)
  - Activation Functions (10 mins)
  - Hands-on (15 mins)

- **Word Embeddings**
  - SVD to Word2Vec (45 mins)
  - PyTorch Data Preparation (60 mins)
  - Word2Vec from Scratch (60 mins)

© 2018 National University of Singapore. All Rights Reserved

Deep Learning Foundations

Recap and activation functions

© 2018 National University of Singapore. All Rights Reserved

Recap: XOR with PyTorch

```
import torch
from torch import nn

X = xor_input = torch.tensor([[0,0], [0,1], [1,0], [1,1]]).float().to(device)
Y = xor_output = torch.tensor([[0], [1], [1], [0]]).float().to(device)

# Define the shape of the weight vector.
num_data, input_dim = X.shape
hidden_dim = 5
output_dim = len(Y)

# Initialize the network as an nn.Sequential.
model = nn.Sequential(
          nn.Linear(input_dim, hidden_dim),
          nn.Sigmoid(),
          nn.Linear(hidden_dim, output_dim),
          nn.Sigmoid()
        ).to(device)

# Declare the loss function as an nn.Module.
criterion = nn.MSELoss()
```

© 2018 National University of Singapore. All Rights Reserved

Recap: XOR with PyTorch

```
num_epochs = 10000
learning_rate = 0.3

# Initialize the optimizer that'll help us with the parameters updates.
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

losses = []
for epoch_n in tqdm(range(num_epochs)):
    # When we move on in this epoch, we should forget the .grad tensors
    # in the previous epoch since we already made the updates.
    optimizer.zero_grad()

    # Forward propagation.
    predictions = model(X)
    # Compute Loss
    loss = criterion(predictions, Y)
    # Keep track of the losses.
    losses.append(loss.item())

    # Backpropagation.
    loss.backward()

    # The step() function will update the parameters in the models that
    # has the .grad tensors respectively.
    optimizer.step()
```

Forward propagation magic!!!

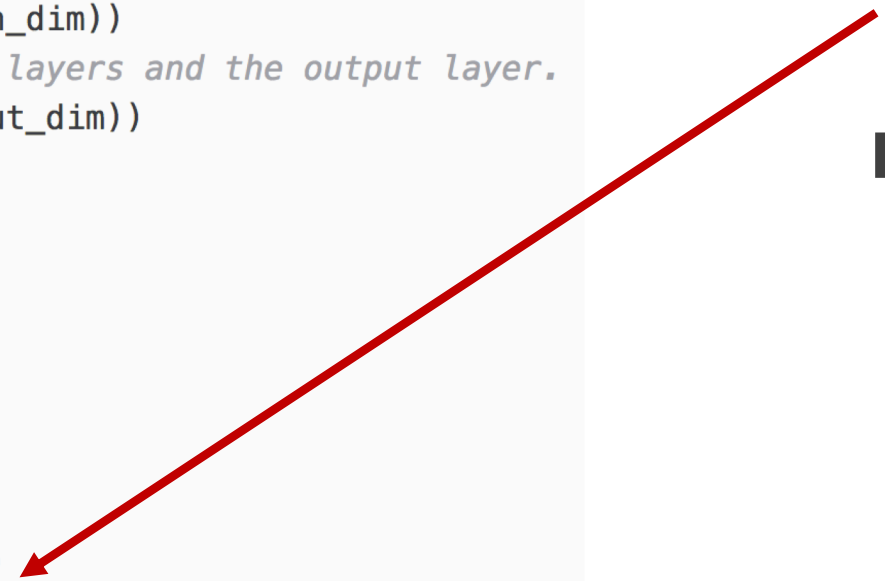
Recap: XOR with PyTorch

```
num_epochs = 10000
learning_rate = 0.3

# Initialize the optimizer that'll help us with the parameters updates.
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

losses = []
for epoch_n in tqdm(range(num_epochs)):
    # When we move on in this epoch, we should forget the .grad tensors
    # in the previous epoch since we already made the updates.
    optimizer.zero_grad()

    # Forward propagation.
    predictions = model(X)
    # Compute Loss
    loss = criterion(predictions, Y)
    # Keep track of the losses.
    losses.append(loss.item())

    # Backpropagation.
    loss.backward()

    # The step() function will update the parameters in the models that
    # has the .grad tensors respectively.
    optimizer.step()
```

Forward propagation magic!!!

Recap: XOR from Scratch

```
3  def sigmoid(x): # Returns values that sums to one.
4      return 1 / (1 + np.exp(-x))
5
6  X = xor_input = np.array([[0,0], [0,1], [1,0], [1,1]])
7  Y = xor_output = np.array([[0,1,1,0]]).T
8
9  # Define the shape of the weight vector.
10 num_data, input_dim = X.shape
11 hidden_dim = 5
12 output_dim = len(Y.T)
13
14 # Initialize weights between the input layers and the hidden layer.
15 W1 = np.random.random((input_dim, hidden_dim))
16 # Initialize weights between the hidden layers and the output layer.
17 W2 = np.random.random((hidden_dim, output_dim))
18
19 # Initialize weigh
20 num_epochs = 5000
21 learning_rate = 0.15
22
23 for epoch_n in range(num_epochs):
24     layer0 = X
25     # Inside the perceptron, Step 2.
26     layer1 = sigmoid(np.dot(layer0, W1))
27     layer2 = sigmoid(np.dot(layer1, W2))
```



Manually doing matrix multiplication

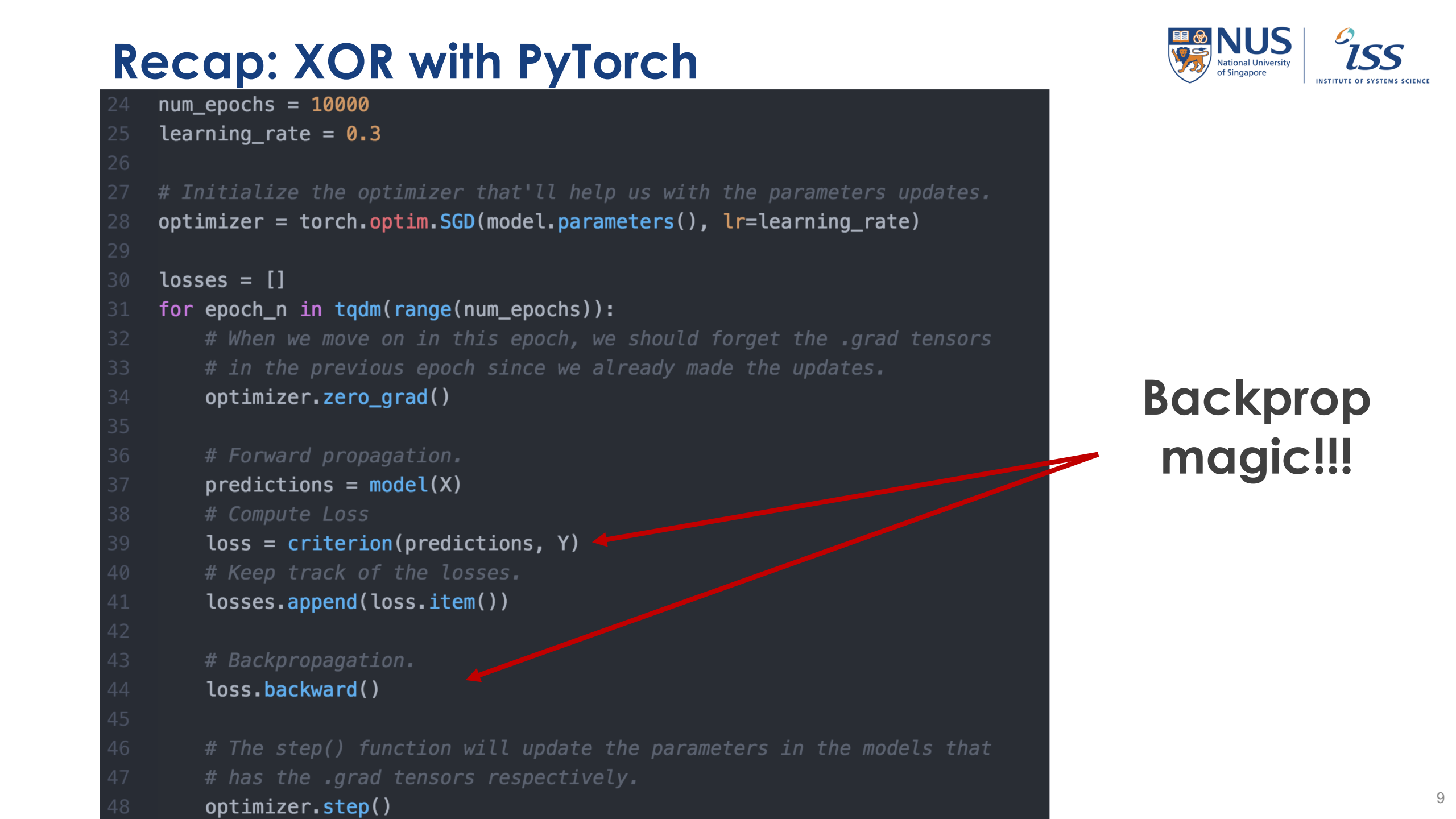
Recap: XOR with PyTorch (BTS)

```
2  def sigmoid(x): # Returns values that sums to one.
3      return 1 / (1 + torch.exp(-x))
4
5  X = xor_input = torch.tensor([[0,0], [0,1], [1,0], [1,1]]).float().to(device)
6  Y = xor_output = torch.tensor([[0], [1], [1], [0]]).float().to(device)
7
8  # Define the shape of the weight vector.
9  num_data, input_dim = X.shape
10 hidden_dim = 5
11 output_dim = len(Y)
12 # When we initialize tensors that needs updating, we use `require_grad=True`
13 # for autograd to kick in later on.
14 W1 = torch.randn(input_dim, hidden_dim, requires_grad=True).to(device)
15 W2 = torch.randn(hidden_dim, output_dim, requires_grad=True).to(device)
16
17 num_epochs = 10000
18 learning_rate = 0.3
19
20 for epoch_n in tqdm(range(num_epochs)):
21     layer0 = X
22     # See https://pytorch.org/docs/stable/torch.html#torch.mm
23     # Use the torch.tensor.mm() instead of np.dot()
24     layer1 = sigmoid(X.mm(W1))
25     layer2 = sigmoid(layer1.mm(W2))
```

PyTorch is actually doing matrix multiplication Behind-The-Scene (BTS)

Recap: XOR with PyTorch

```
num_epochs = 10000
learning_rate = 0.3

# Initialize the optimizer that'll help us with the parameters updates.
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

losses = []
for epoch_n in tqdm(range(num_epochs)):
    # When we move on in this epoch, we should forget the .grad tensors
    # in the previous epoch since we already made the updates.
    optimizer.zero_grad()

    # Forward propagation.
    predictions = model(X)
    # Compute Loss
    loss = criterion(predictions, Y)
    # Keep track of the losses.
    losses.append(loss.item())

    # Backpropagation.
    loss.backward()

    # The step() function will update the parameters in the models that
    # has the .grad tensors respectively.
    optimizer.step()
```

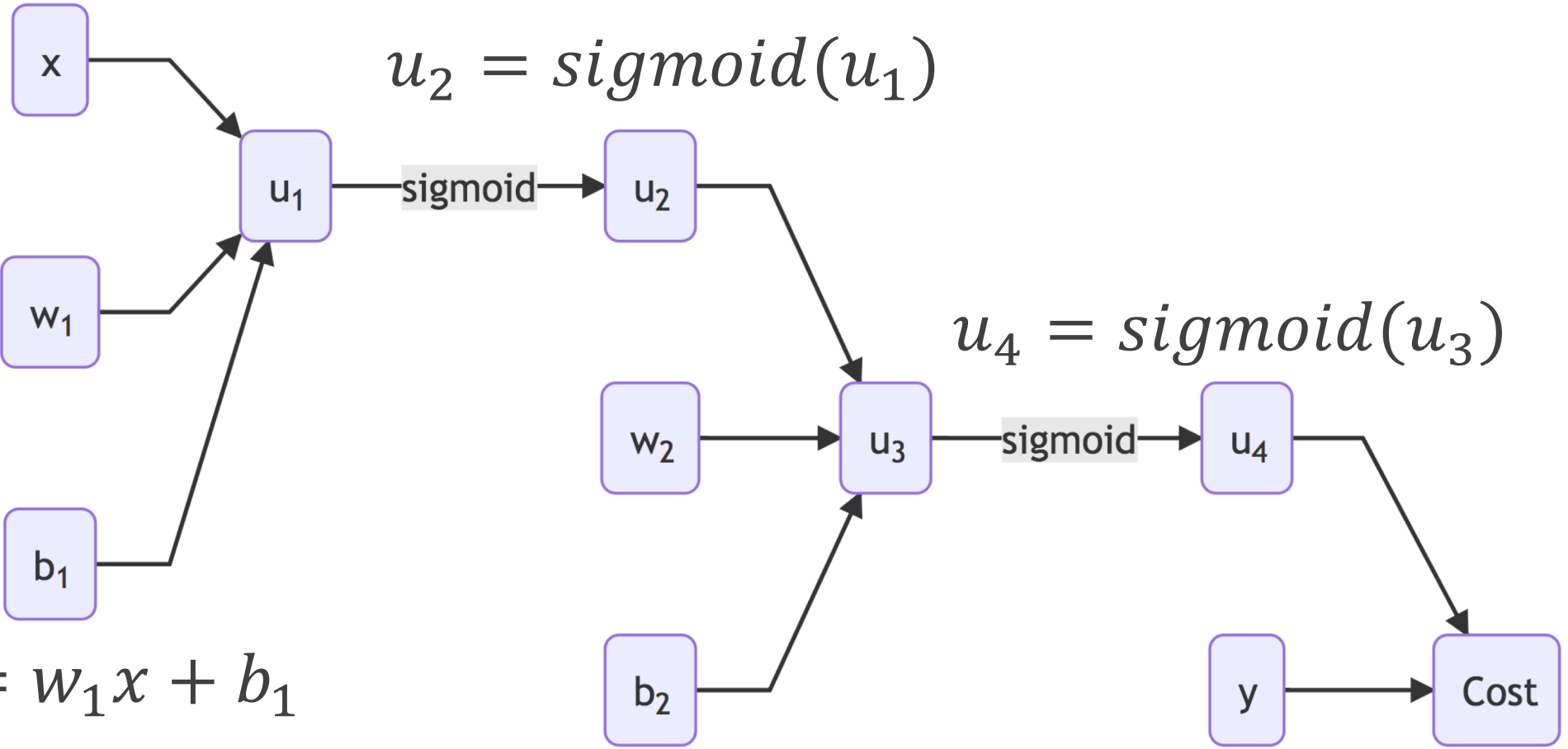


Recap: XOR from Scratch

```
for epoch_n in range(num_epochs):
    layer0 = X
    # Forward propagation.

    # Inside the perceptron, Step 2.
    layer1 = sigmoid(np.dot(layer0, W1))
    layer2 = sigmoid(np.dot(layer1, W2))

    # Back propagation (Y -> layer2)
    # How much did we miss in the predictions?
    cost_error = mse(layer2, Y)

    # In what direction is the target value?
    # Were we really close? If so, don't change too much.
    layer2_error = mse_derivative(layer2, Y)
    layer2_delta = layer2_error *  sigmoid_derivative(layer2)

    # Back propagation (layer2 -> layer1)
    # How much did each layer1 value contribute to the layer2 error (according to the weights)?
    layer1_error = np.dot(layer2_delta, W2.T)
    layer1_delta = layer1_error * sigmoid_derivative(layer1)

    # update weights
    W2 += - learning_rate * np.dot(layer1.T, layer2_delta)
    W1 += - learning_rate * np.dot(layer0.T, layer1_delta)
    #print(np.dot(layer0.T, layer1_delta))
    #print(epoch_n, list((layer2)))

    # Log the loss value as we proceed through the epochs.
    losses.append(cost_error)
    #print(cost_delta)
```

Backprop pain!!!

Recap: Derivatives of a Multi-Layered Perceptron



$u_2 = sigmoid(u_1)$

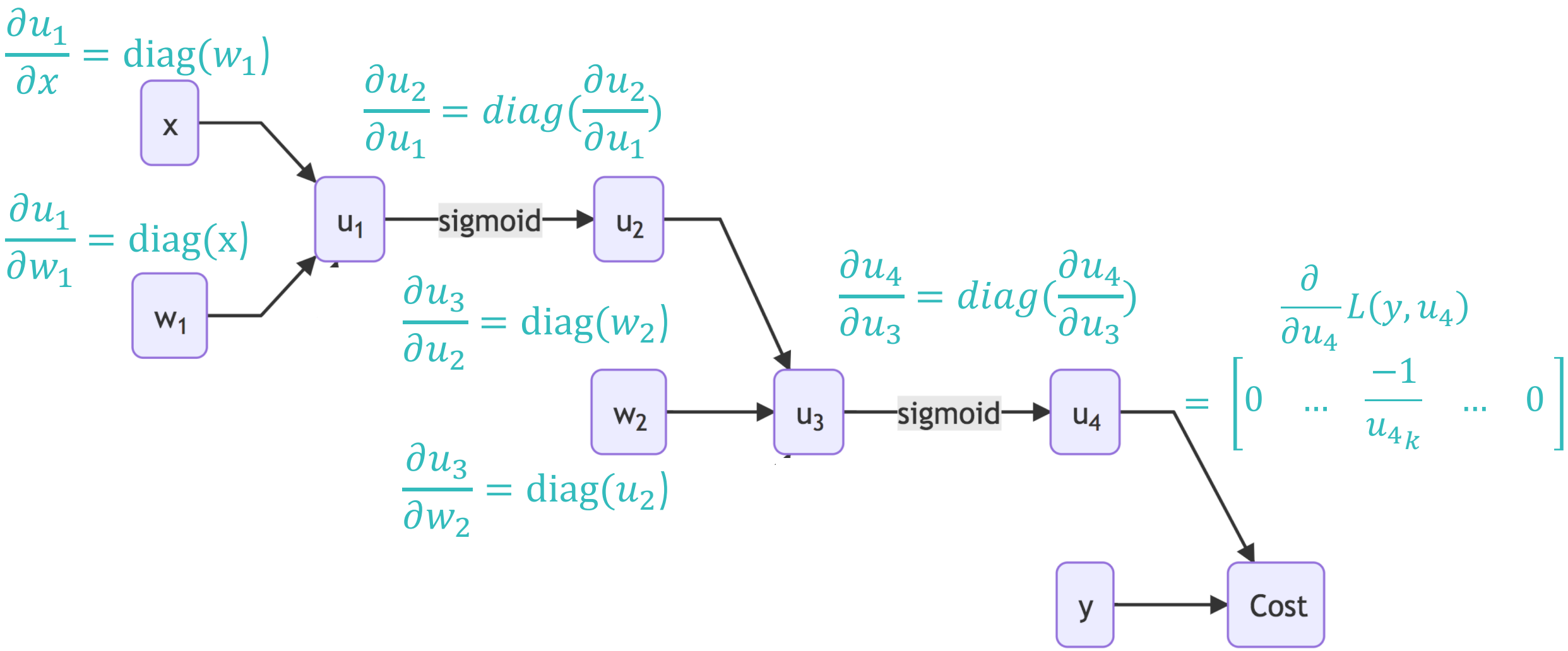
$u_4 = sigmoid(u_3)$

$u_1 = w_1 x + b_1$

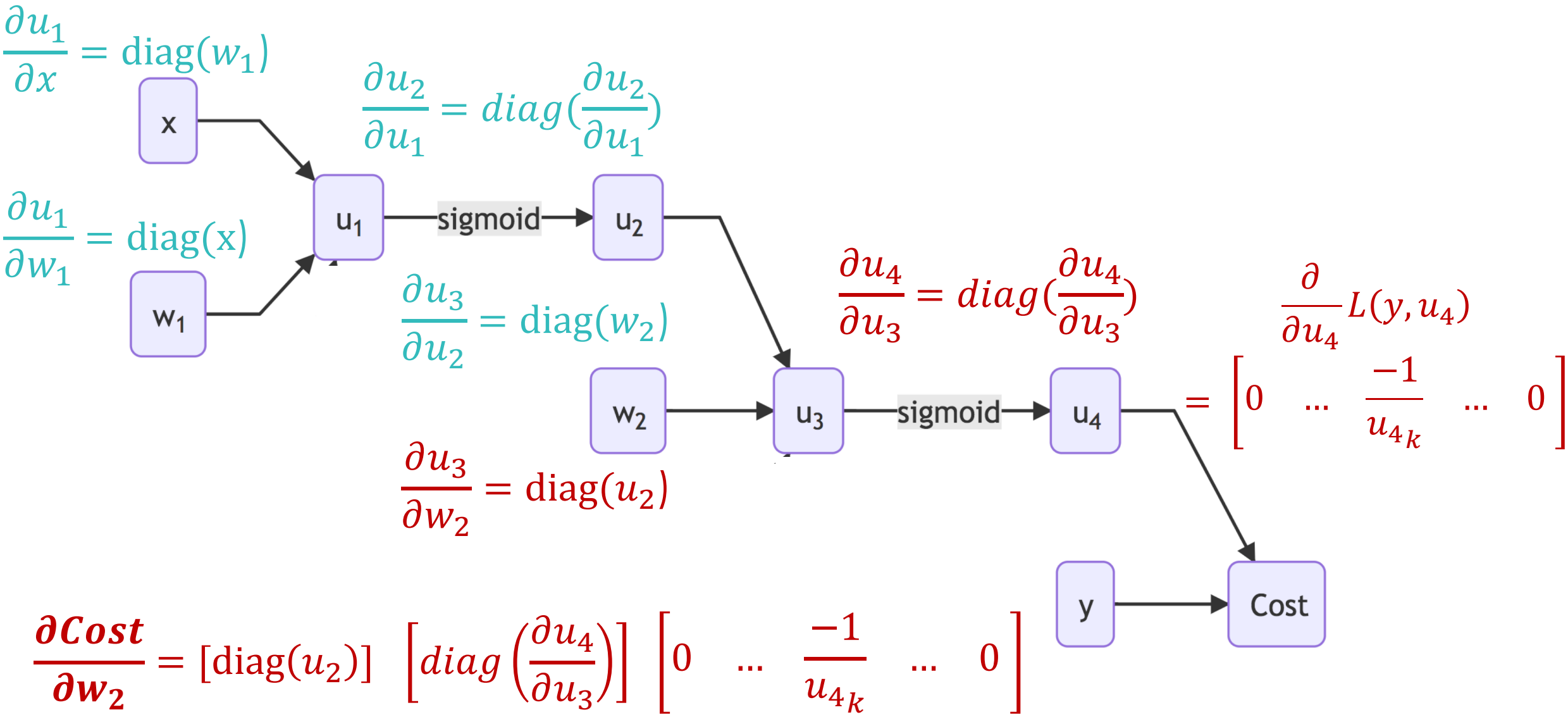
$u_3 = w_2 u_2 + b_2$

$cost = L(y, u_4)$

Recap: Derivatives of a Multi-Layered Perceptron

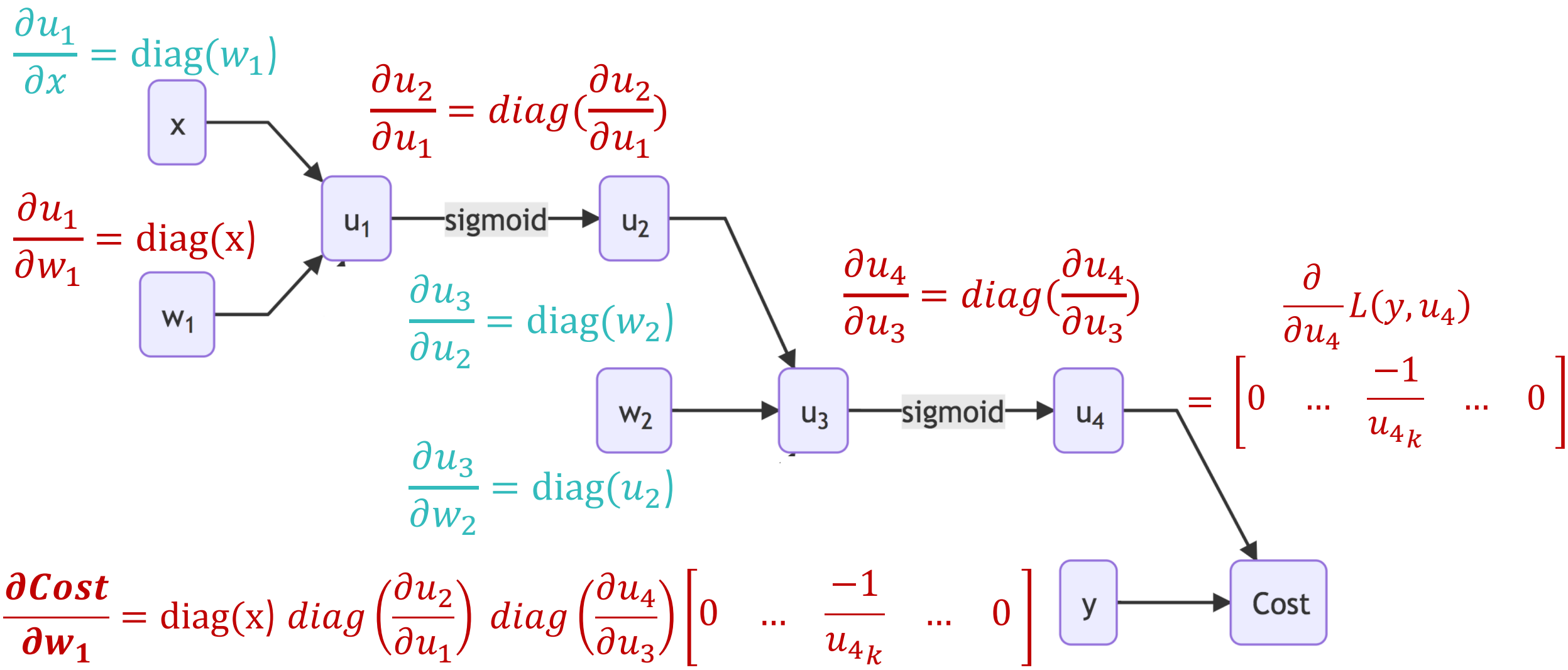


Recap: Derivatives of a Multi-Layered Perceptron



$$\frac{\partial Cost}{\partial w_2} = [\text{diag}(u_2)] \left[diag\left(\frac{\partial u_4}{\partial u_3}\right)\right] \left[0 \quad \dots \quad \frac{-1}{u_{4_k}} \quad \dots \quad 0\right]$$

Recap: Derivatives of a Multi-Layered Perceptron



Recap: Derivatives of a Multi-Layered Perceptron

$$\frac{\partial Cost}{\partial w_1} = \text{diag(x)}\ diag\left(\frac{\partial u_2}{\partial u_1}\right)\ diag\left(\frac{\partial u_4}{\partial u_3}\right)\begin{bmatrix} 0 & \dots & \frac{-1}{u_{4_k}} & \dots & 0 \end{bmatrix}$$

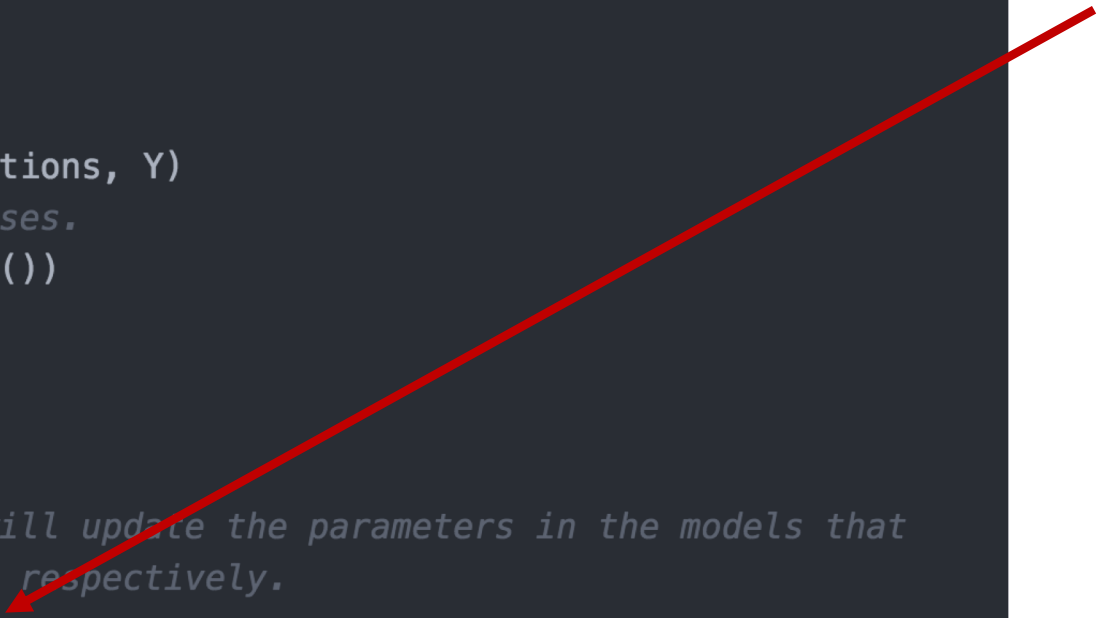
$$\frac{\partial Cost}{\partial w_2} = \text{diag}(u_2)\ diag\left(\frac{\partial u_4}{\partial u_3}\right)\begin{bmatrix} 0 & \dots & \frac{-1}{u_{4_k}} & \dots & 0 \end{bmatrix}$$

$$w_2 \mathrel{+}= -lr * \frac{\partial Cost}{\partial w_2}$$

$$w_1 \mathrel{+}= -lr * \frac{\partial Cost}{\partial w_1}$$

Recap: XOR with PyTorch

```
num_epochs = 10000
learning_rate = 0.3

# Initialize the optimizer that'll help us with the parameters updates.
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

losses = []
for epoch_n in tqdm(range(num_epochs)):
    # When we move on in this epoch, we should forget the .grad tensors
    # in the previous epoch since we already made the updates.
    optimizer.zero_grad()

    # Forward propagation.
    predictions = model(X)
    # Compute Loss
    loss = criterion(predictions, Y)
    # Keep track of the losses.
    losses.append(loss.item())

    # Backpropagation.
    loss.backward()

    # The step() function will update the parameters in the models that
    # has the .grad tensors respectively.
    optimizer.step()
```



Optimizer weights updating magic!!!

Recap: XOR from Scratch

```
for epoch_n in range(num_epochs):
    layer0 = X
    # Forward propagation.

    # Inside the perceptron, Step 2.
    layer1 = sigmoid(np.dot(layer0, W1))
    layer2 = sigmoid(np.dot(layer1, W2))

    # Back propagation (Y -> layer2)
    # How much did we miss in the predictions?
    cost_error = mse(layer2, Y)

    # In what direction is the target value?
    # Were we really close? If so, don't change too much.
    layer2_error = mse_derivative(layer2, Y)
    layer2_delta = layer2_error *  sigmoid_derivative(layer2)

    # Back propagation (layer2 -> layer1)
    # How much did each layer1 value contribute to the layer2 error (according to the weights)?
    layer1_error = np.dot(layer2_delta, W2.T)
    layer1_delta = layer1_error * sigmoid_derivative(layer1)

    # update weights
    W2 += - learning_rate * np.dot(layer1.T, layer2_delta)
    W1 += - learning_rate * np.dot(layer0.T, layer1_delta)
    #print(np.dot(layer0.T, layer1_delta))
    #print(epoch_n, list((layer2)))

    # Log the loss value as we proceed through the epochs.
    losses.append(cost_error)
    #print(cost_delta)
```

Painful weights updates

Recap: XOR with PyTorch (BTS)

```
for epoch_n in tqdm(range(num_epochs)):
    layer0 = X
    layer1 = sigmoid(X.mm(W1))
    layer2 = sigmoid(layer1.mm(W2))

    # Loss is a Tensor of torch.Size([]), so and loss.item() is a scalar/float.
    # Try printing `print(loss.shape)` to confirm the above.
    loss = mse(layer2, Y)
    # Keep track of the losses.
    losses.append(loss.item())

    # The `loss.backward()` will compute the gradient of loss w.r.t. all
    # tensors that has `requires_grad=True`.
    # After this, W1.grad and W2.grad will hold the gradients of
    # the loss w.r.t. to W1 and W2 respectively.
    loss.backward()

    # Now we have the backpropagated gradients, we want to update the weights.
    # Whenever you perform tensor operations on tensors that has `requires_grad=True`,
    # pytorch will try to build computation graph. For now, we only need to
    # force the updates of our weights without forming more computation graph,
    # so we use the no_grad() context manager:
    with torch.no_grad():
        W1 += -learning_rate * W1.grad
        W2 += -learning_rate * W2.grad
```

PyTorch Optimizer weights updating Behind-The-Scene (BTS)

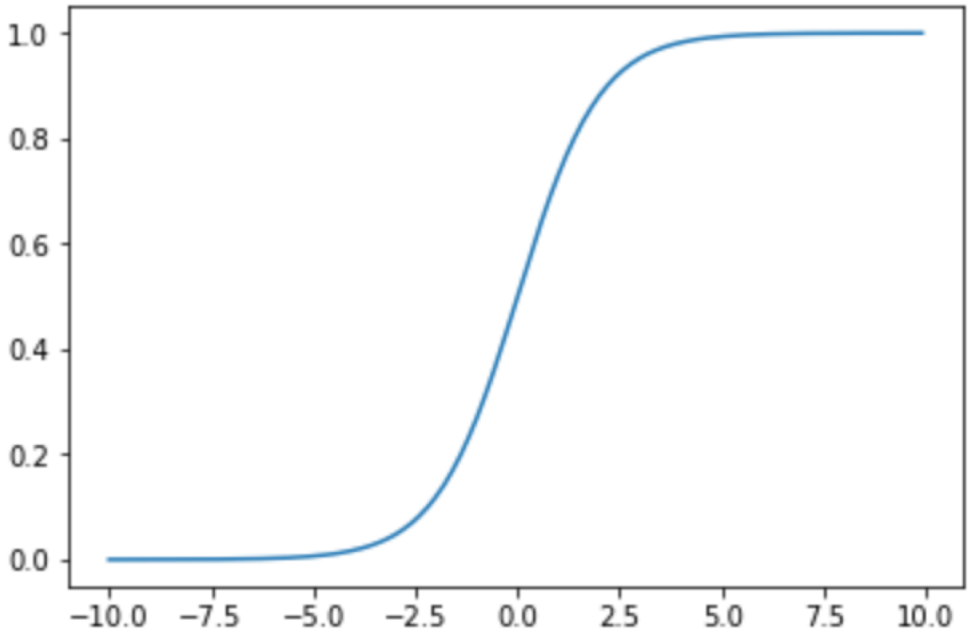
Activation Functions

Sigmoid, S-Shape, Rectified Activations

© 2018 National University of Singapore. All Rights Reserved

Activation Function (Sigmoid)

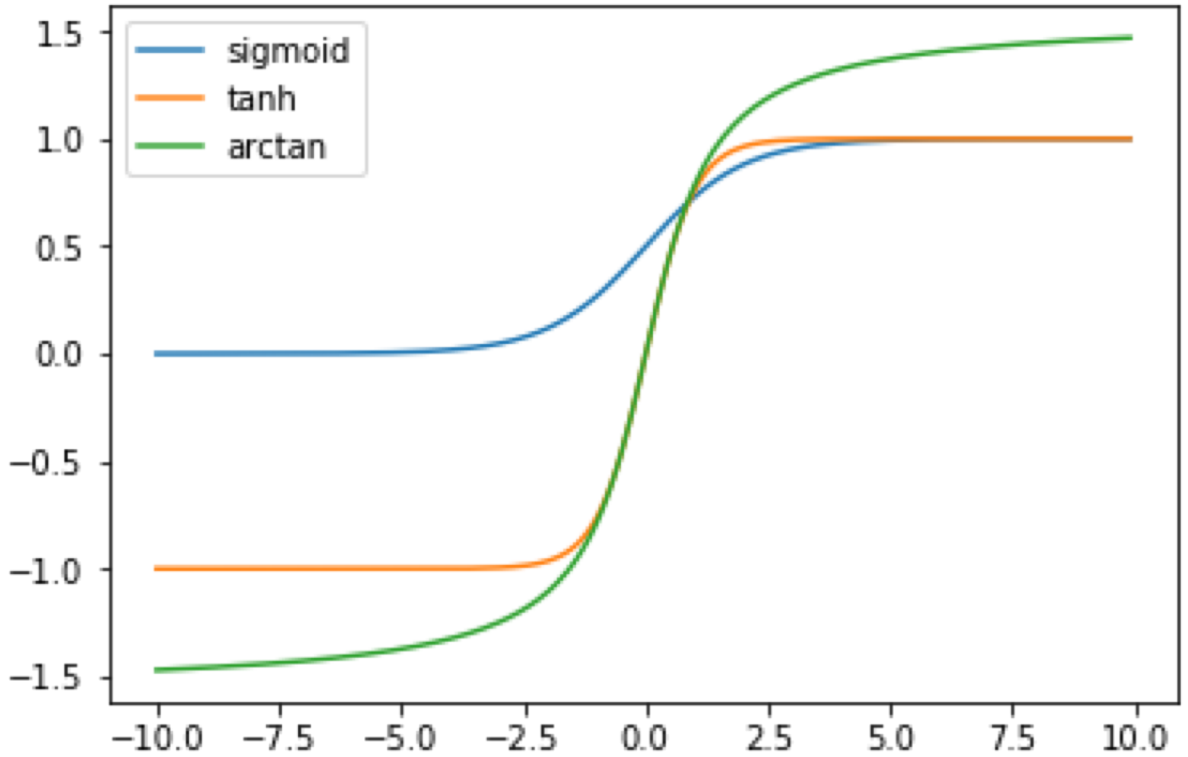
```
3   import numpy as np
4   import matplotlib.pyplot as plt
5
6   def sigmoid(x):
7       return 1/(1+np.exp(-x))
8
9   # Generate points from -10 to +10,
10  # in steps of 0.1
11  x = np.arange(-10, 10, 0.1)
12  y = sigmoid(x)
13
14  # Plot the graph.
15  plt.plot(x, y)
16  plt.show()
```



© 2018 National University of Singapore. All Rights Reserved

Activation Function (S-Shape Activations)

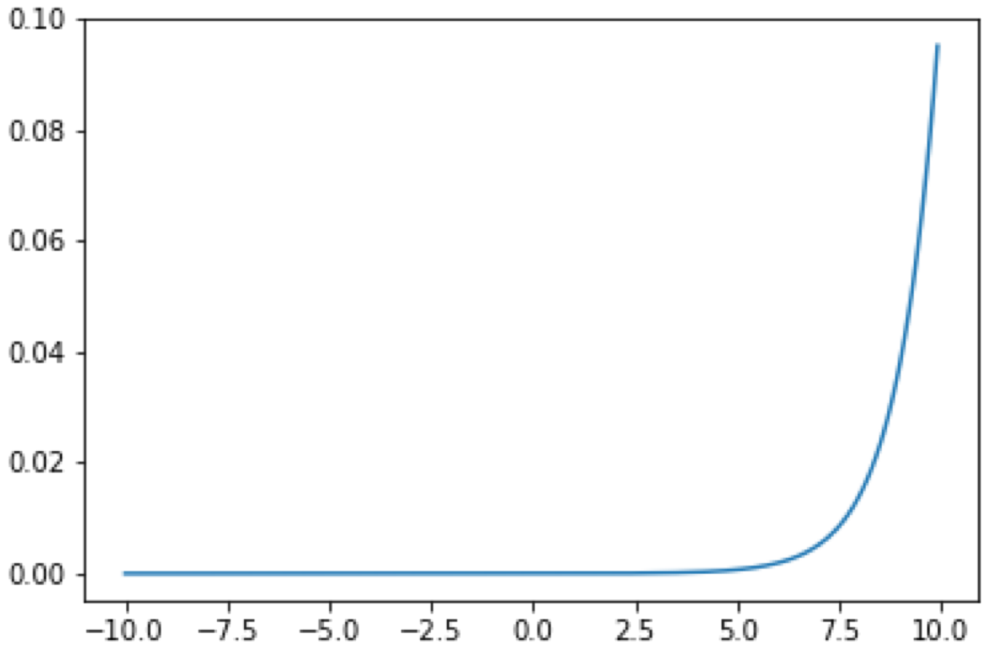
```
6   def sigmoid(x):
7       return 1 / (1+np.exp(-x))
8
9   def tanh(x):
10      return np.tanh(x)
11
12  def arctan(x):
13      return np.arctan(x)
14
15  x = np.arange(-10, 10, 0.1)
16  y1 = sigmoid(x)
17  y2 = tanh(x)
18  y3 = arctan(x)
19
20
21  plt.plot(x,y1, label='sigmoid')
22  plt.plot(x,y2, label='tanh')
23  plt.plot(x,y3, label='arctan')
24  plt.legend(loc='upper left')
25  plt.show()
```



© 2018 National University of Singapore. All Rights Reserved

Activation Function (Softmax)

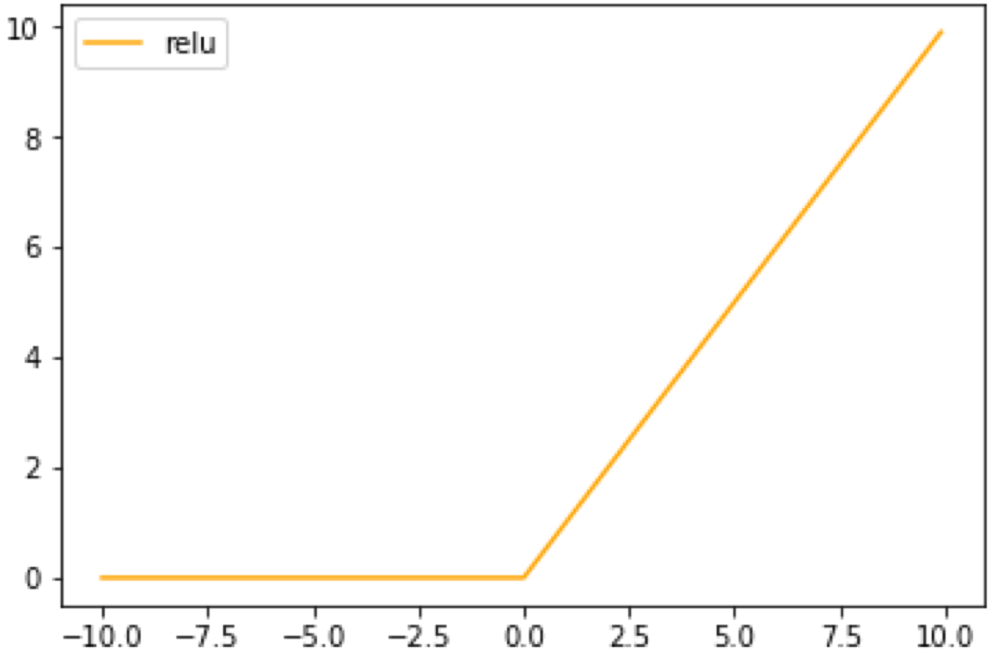
```
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  def softmax(x):
7      return np.exp(x) / np.sum(np.exp(x), axis=0)
8
9  x = np.arange(-10, 10, 0.1)
10 y = softmax(x)
11
12 plt.plot(x,y)
13 plt.show()
```



© 2018 National University of Singapore. All Rights Reserved

Activation Function (ReLU)

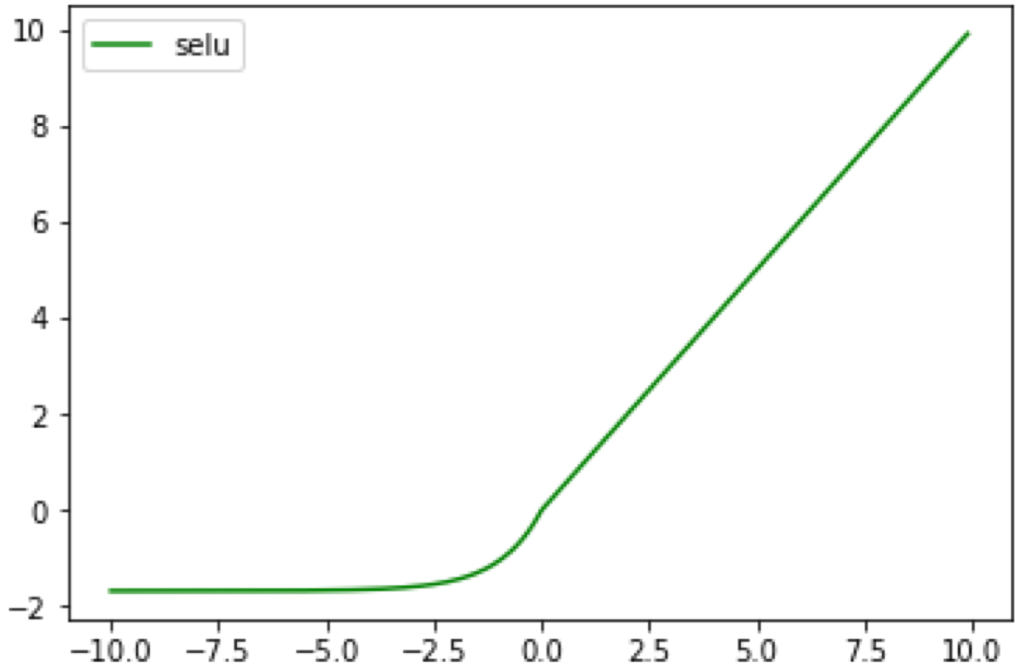
```
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  def relu(x):
7      return x * (x > 0)
8
9  y2 = relu(x)
10
11 plt.plot(x,y2, label='relu', color='orange')
12 plt.legend(loc='upper left')
13 plt.show()
```



© 2018 National University of Singapore. All Rights Reserved

Activation Functions (SELU)

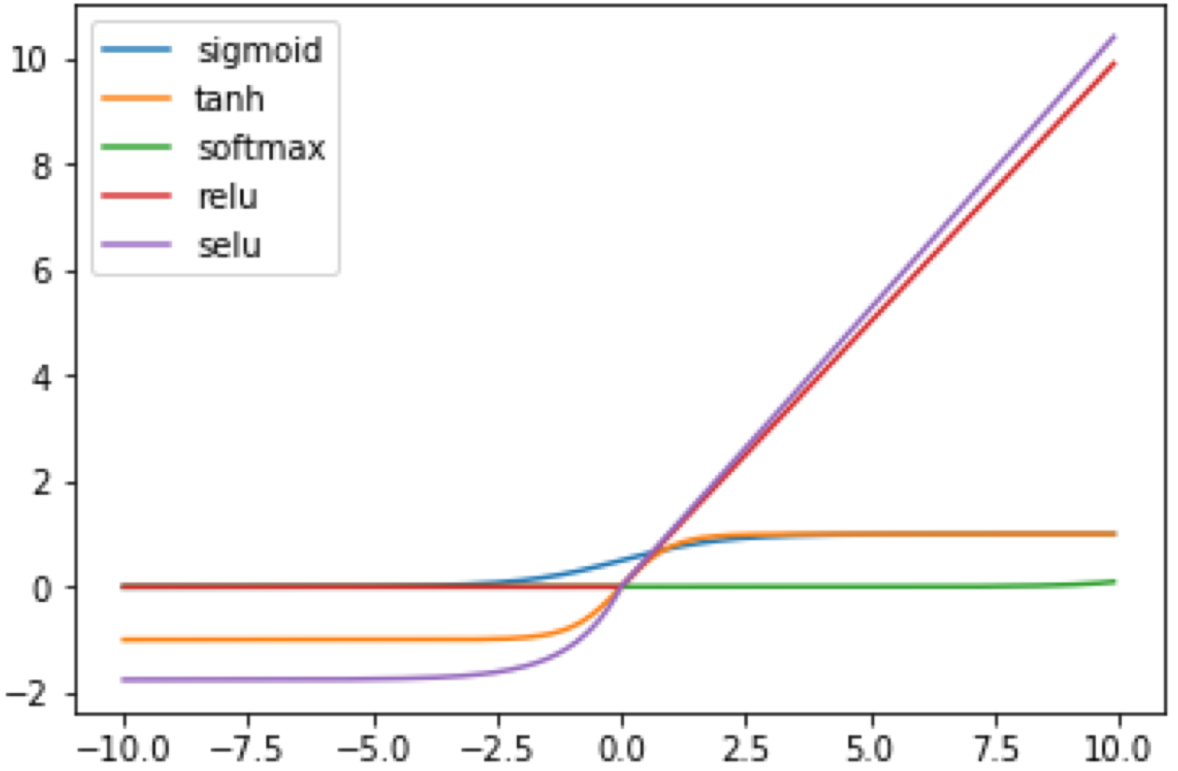
```
def selu(x):
    alpha = 1.6732632423543772848170429916717
    scale = 1.0507009873554804934193349852946
    return (np.maximum(0, x) +
            np.minimum(0, alpha*(np.exp(x) -1)))

y3 = selu(x)

plt.plot(x,y3, label='selu', color='green')
plt.legend(loc='upper left')
plt.show()
```



© 2018 National University of Singapore. All Rights Reserved

Activation Functions with PyTorch

```
3   from torch import nn, tensor
4
5   x = tensor(np.arange(-10, 10, 0.1))
6
7   a1 = nn.Sigmoid()
8   a2 = nn.Tanh()
9   a3 = nn.Softmax()
10  a4 = nn.ReLU()
11  a5 = nn.SELU()
12
13  y1, y2, y3 = a1(x), a2(x), a3(x)
14  y4, y5 = a4(x), a5(x)
15
16  plt.plot(x,y1, label='sigmoid')
17  plt.plot(x,y2, label='tanh')
18  plt.plot(x,y3, label='softmax')
19  plt.plot(x,y4, label='relu')
20  plt.plot(x,y5, label='selu')
21  plt.legend(loc='upper left')
22  plt.show()
```



© 2018 National University of Singapore. All Rights Reserved

Hands-on: Unmagical PyTorch

Possibly XOR one more time =)

© 2018 National University of Singapore. All Rights Reserved

Environment Setup

Open Anaconda Navigator.

Go to the PyTorch installation page, copy the command as per configuration: https://pytorch.org/get-started/locally/

Fire up the terminal in Anaconda Navigator.

Start a Jupyter Notebook.

Download http://bit.ly/ANLP-Session2-Empty-

Import the .ipynb to the Jupyter Notebook

© 2018 National University of Singapore. All Rights Reserved

Word Embeddings

© 2018 National University of Singapore. All Rights Reserved

Word Embeddings

"***You shall know a word by the company it keeps…***"
– John R. Firth (1957)

"We propose a unified NN architecture by trying to avoid task-specific engineering therefore disregarding a lot of prior knowledge"
– Collobert and Weston (2011)

"Context-predicting models known as embeddings are the new kids on the distributional semantics block… The result, to our own surprise, show that the buzz is fully justified."
– Baroni et al. (2014)

© 2018 National University of Singapore. All Rights Reserved

Classic NLP: Feature Engineering

- **Count-based vectors are**
  - e.g. TF-IDF, PPMI
  - long (|V| > 100,000)
  - sparse (lots of zero)

- **Vector compression** (aka ***dimensionality reduction***)
  - shorter vectors easier to use as features in machine learning
  - ***compression*** use to make vectors short and dense, e.g. Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF)

© 2018 National University of Singapore. All Rights Reserved

Term Frequency – Inverse Document Frequency

```
sent0 = "The quick brown fox jumps over the lazy brown dog ."

sent1 = "Mr brown jumps over the lazy fox ."
```

	brown	dog	fox	jumps	lazy	mr	over	quick	the
sent0	0.500	0.351	0.250	0.250	0.250	0.000	0.250	0.351	0.500
sent1	0.354	0.000	0.354	0.354	0.354	0.497	0.354	0.000	0.354

© 2018 National University of Singapore. All Rights Reserved

Term Frequency – Inverse Document Frequency

```
sent0 = "The quick brown fox jumps over the lazy brown dog ."

sent1 = "Mr brown jumps over the lazy fox ."
```

	brown	dog	fox	jumps	lazy	mr	over	quick	the
sent0	0.500	0.351	0.250	0.250	0.250	0.000	0.250	0.351	0.500
sent1	0.354	0.000	0.354	0.354	0.354	0.497	0.354	0.000	0.354

v(‘dog’)

© 2018 National University of Singapore. All Rights Reserved

Pointwise Mutual Information

		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
		mr	brown	jumps	over	fox
$i = 1$	mr	0	1	1	0	0
$i = 2$	brown	0	0	1	1	2
$i = 3$	jumps	1	0	0	2	1
$i = 4$	over	2	1	0	0	1
$i = 5$	fox	0	0	1	0	0

Matrix F with
- W rows (words)
- C columns (context)

© 2018 National University of Singapore. All Rights Reserved

Pointwise Mutual Information

		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
		mr	brown	jumps	over	fox
$i = 1$	mr	0	1	1	0	0
$i = 2$	brown	0	0	1	1	2
$i = 3$	jumps	1	0	0	2	1
$i = 4$	over	2	1	0	0	1
$i = 5$	fox	0	0	1	0	0

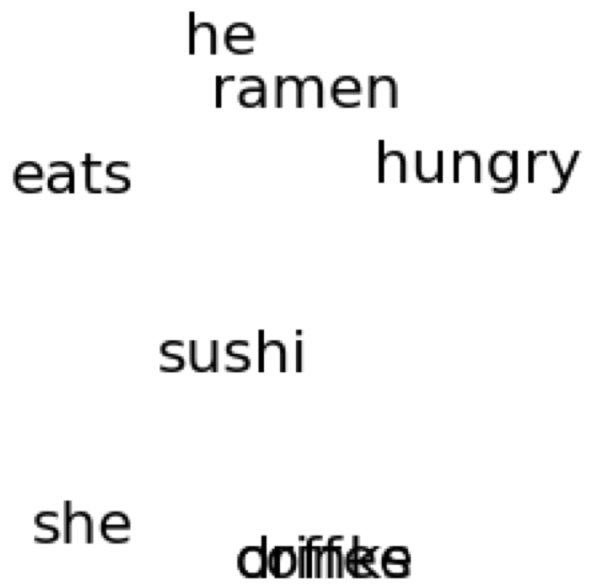
Matrix F with
- W rows (words)
- C columns (context)

v('fox')

© 2018 National University of Singapore. All Rights Reserved

Dimensionality Reduction (SVD)

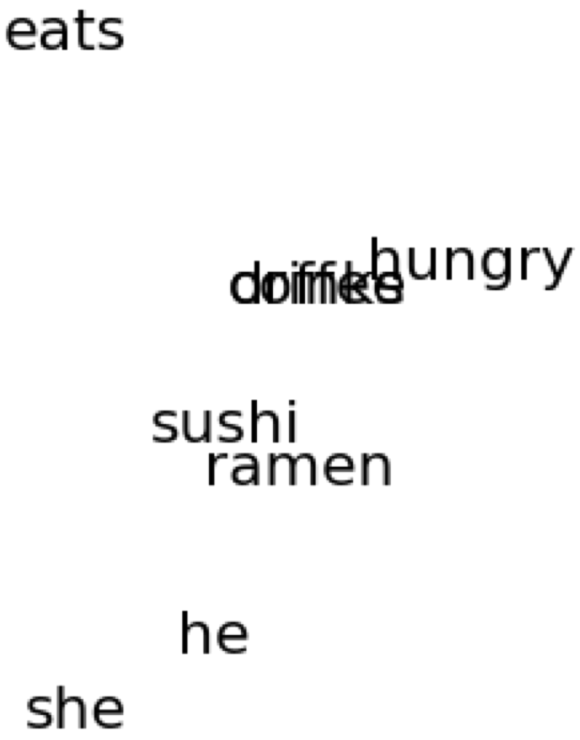
```
corpus = ['he eats ramen', 'she eats sushi',
          'he hungry', 'she drinks coffee']

# Get the standard term-doc matrix
count_model = CountVectorizer(ngram_range=(1,1))
X = count_model.fit_transform(corpus)
# Multiplying the term-doc matrix with itself
# produces the co-occurence matrix.
X_cooc = (X.T * X)
X_cooc.setdiag(0)   # set co-occurence with self to 0.

U, s, Vh = np.linalg.svd(X_cooc.todense(),
                         full_matrices=False)

words = sorted(count_model.vocabulary_,
               key=count_model.vocabulary_.get)

# Plot pretty words.
for i in range(len(words)):
    plt.text(U[i,0], U[i,1], words[i], fontsize=22)
plt.axis('off')
plt.show()
```



© 2018 National University of Singapore. All Rights Reserved

Dimensionality Reduction (SVD)

```
corpus = ['he eats ramen', 'she eats sushi',
          'he hungry', 'she drinks coffee']

# Get the standard term-doc matrix
count_model = CountVectorizer(ngram_range=(1,1))
X = count_model.fit_transform(corpus)
# Multiplying the term-doc matrix with itself
# produces the co-occurence matrix.
X_cooc = (X.T * X)
X_cooc.setdiag(0)   # set co-occurence with self to 0.

U, s, Vh = np.linalg.svd(X_cooc.todense(),
                         full_matrices=False)

words = sorted(count_model.vocabulary_,
               key=count_model.vocabulary_.get)

# Plot pretty words.
for i in range(len(words)):
    plt.text(U[i,0], U[i,2], words[i], fontsize=22)
plt.axis('off')
plt.show()
```



© 2018 National University of Singapore. All Rights Reserved

Single Value Decomposition (SVD)

- Computational cost scales quadratically,

- O(mn^2) for n x m matrix
  - Even if we can flip the n/m easily, if we have lots of words/documents, it won't help much.

- If there's new words or documents, SVD has to be re-computed from scratch.

© 2018 National University of Singapore. All Rights Reserved

Count-based Vectors

tokenization
annotation
tagging
parsing
feature selection
⋮ cluster texts by date/author/discourse context/...
↓ ↙

Matrix type		Weighting		Dimensionality reduction		Vector comparison
word × document		probabilities		LSA		Euclidean
word × word		length normalization		PLSA		Cosine
word × search proximity	×	TF-IDF	×	LDA	×	Dice
adj. × modified noun		PMI		PCA		Jaccard
word × dependency rel.		Positive PMI		IS		KL
verb × arguments		PPMI with discounting		DCA		KL with skew
⋮		⋮		⋮		⋮

(Nearly the full cross-product to explore; only a handful of the combinations are ruled out mathematically, and the literature contains relatively little guidance.)

Potts (2013)

© 2018 National University of Singapore. All Rights Reserved

Don't Count, Predict!

	rg	ws	wss	wsr	men	toefl	ap	esslli	battig	up	mcrae	an	ansyn	ansem
best setup on each task														
cnt	74	62	70	59	72	76	66	84	98	41	27	49	43	60
pre	84	75	**80**	**70**	**80**	91	75	86	**99**	41	28	**68**	**71**	**66**
best setup across tasks														
cnt	70	62	70	57	72	76	64	84	98	37	27	43	41	44
pre	83	73	78	68	**80**	86	71	77	98	41	26	67	69	64
worst setup across tasks														
cnt	11	16	23	4	21	49	24	43	38	-6	-10	1	0	1
pre	74	60	73	48	68	71	65	82	88	33	20	27	40	10
best setup on rg														
cnt	(74)	59	66	52	71	64	64	84	98	37	20	35	42	26
pre	(84)	71	76	64	79	85	72	84	98	39	25	66	70	61
other models														
soa	**86**	**81**	77	62	76	**100**	**79**	**91**	96	**60**	**32**	61	64	61
dm	82	35	60	13	42	77	76	84	94	51	29	NA	NA	NA
cw	48	48	61	38	57	56	58	61	70	28	15	11	12	9

Table 2: Performance of count (cnt), predict (pre), dm and cw models on all tasks. See Section 3 and Table 1 for figures of merit and state-of-the-art results (soa). Since dm has very low coverage of the an* data sets, we do not report its performance there.

Baroni et al. (2014)

© 2018 National University of Singapore. All Rights Reserved

Word Embeddings: Not New, but Different

- Learning representations by back-propagating errors (Rumelhart et al. 1986)
- Neural Probabilistic Language Model (Bengio et al. 2003)
- NLP (almost) from Scratch (Collobert and Weston, 2008)
- Word2Vec (Mikolov et al., 2013)

© 2018 National University of Singapore. All Rights Reserved

One-Hot Encoding (Sparse Representation)

	he	she	eats	drinks	sushi	ramen	hungry	coffee
he	1	0	0	0	0	0	0	0
she	0	1	0	0	0	0	0	0
eats	0	0	1	0	0	0	0	0
drinks	0	0	0	1	0	0	0	0
sushi	0	0	0	0	1	0	0	0
ramen	0	0	0	0	0	1	0	0
hungry	0	0	0	0	0	0	1	0
coffee	0	0	0	0	0	0	0	1

v('drinks')

© 2018 National University of Singapore. All Rights Reserved

Word Embeddings (Dense Representation)

	he	she	eats	drinks	sushi	ramen	hungry	coffee
0	0.1	0.2	-0.4	0.9	0.8	0.1	0.8	-0.8
1	0.2	0.1	-0.3	0.9	0.7	0.2	0.3	-2.1
2	0.2	-1.4	0.3	-0.1	0.1	0.5	0.9	-0.5
3	0.3	-2.0	0.5	-0.5	0.2	0.4	0.1	-0.1
4	0.2	-1.1	0.3	-0.7	-0.6	-0.5	0.3	0.4
5	0.3	-1.2	0.4	-0.9	-0.3	-0.4	-0.6	-0.4

v('drinks')

© 2018 National University of Singapore. All Rights Reserved

Lookup Function

0
0
0
1
0
0
0
0

X

0.1	0.2	-0.4	0.9	0.8	0.1	0.8	-0.8
0.2	0.1	-0.3	0.9	0.7	0.2	0.3	-2.1
0.2	-1.4	0.3	-0.1	0.1	0.5	0.9	-0.5
0.3	-2.0	0.5	-0.5	0.2	0.4	0.1	-0.1
0.2	-1.1	0.3	-0.7	-0.6	-0.5	0.3	0.4
0.3	-1.2	0.4	-0.9	-0.3	-0.4	-0.6	-0.4

0.9
0.9
-0.1
-0.5
-0.7
-0.9

One-Hot Encoding

1 x |V|

Word Embeddings

|V| x d

Input

1 x d

© 2018 National University of Singapore. All Rights Reserved

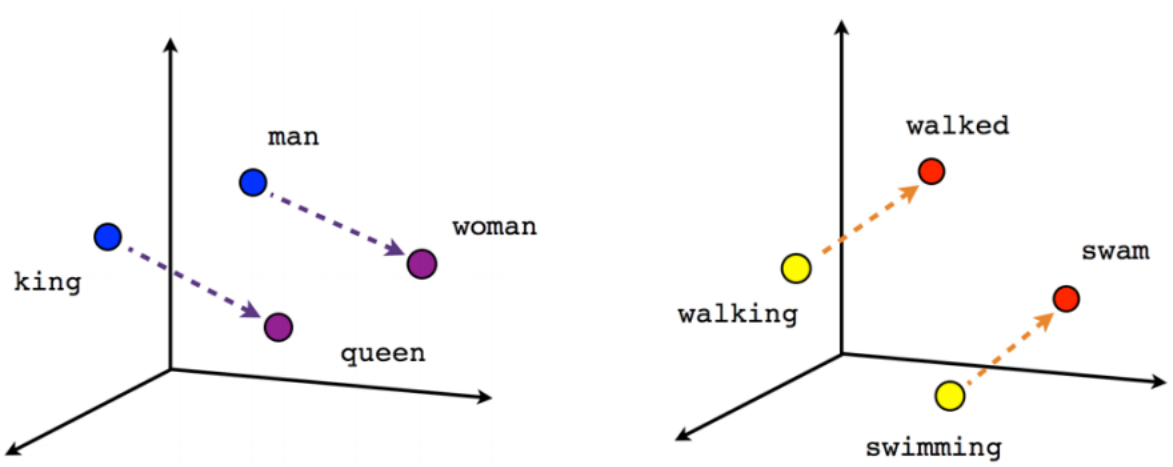
Deep NLP: Featurize and Predict

- **Deep learning can create vectors that are**
  - short (often fixed-sized <2000, decided empirically)
  - dense (most are non-zeros)

- **How might we "featurize" the vectors through some tasks and update the vectors based on gradient descent?**

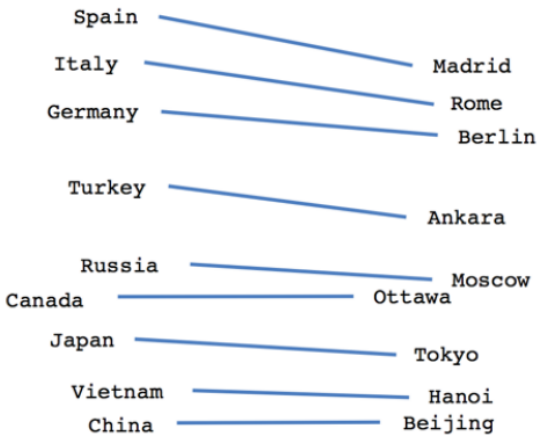
© 2018 National University of Singapore. All Rights Reserved

Word2Vec



Male-Female

Verb tense



Country-Capital

Ingredients

Corpus of text	As large as possible
Annotations	0
Initialize weights (aka Embeddings)	1x per word
Deep Learning Model	1x
Cost Function	Appropriately
GPU	Lotsa of it

© 2018 National University of Singapore. All Rights Reserved

Word2Vec

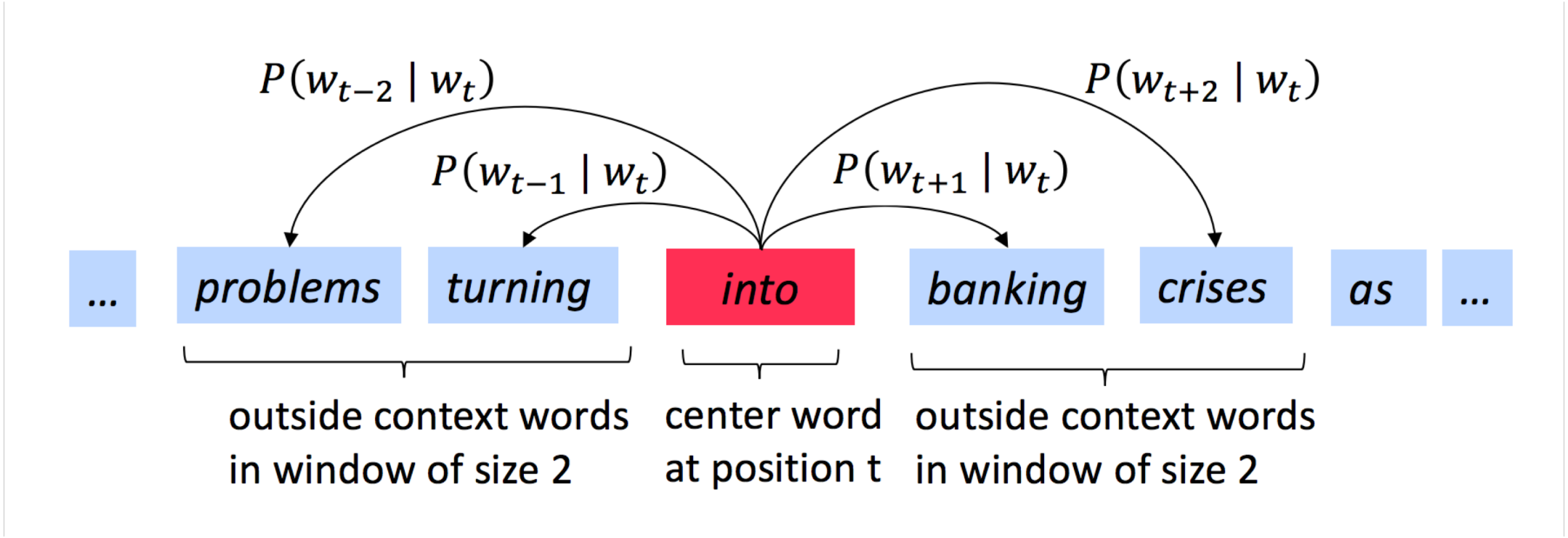
Steps

1. Define task that we want to predict
2. Go through each sentence and create the task's in-/outputs
3. Iterate through task's I/O, put the inputs through the embeddings and models to create predictions
4. Measure cost of the predicted and expected output
5. Update embedding weights accordingly (*backprop)
6. Repeat Step 3-5 until desired.

© 2018 National University of Singapore. All Rights Reserved

Word2Vec (CBOW)

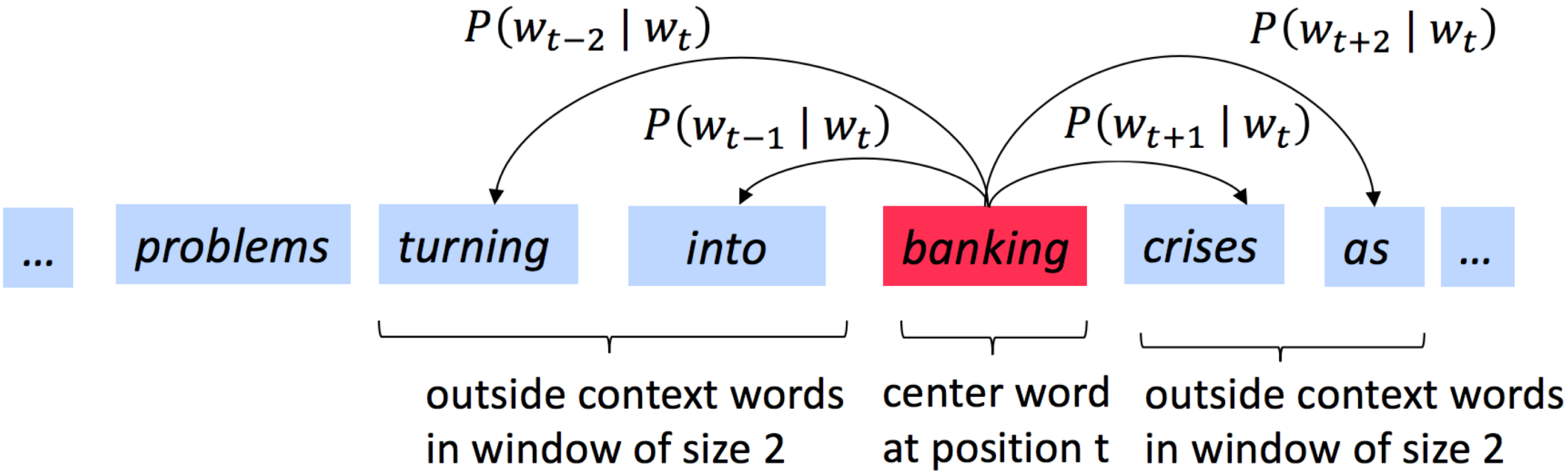
Task: Iterate through each word with a given window; for each word predict the context words within the window



(E.g. from Manning (2018) Stanford cs224n course)

Word2Vec (CBOW)

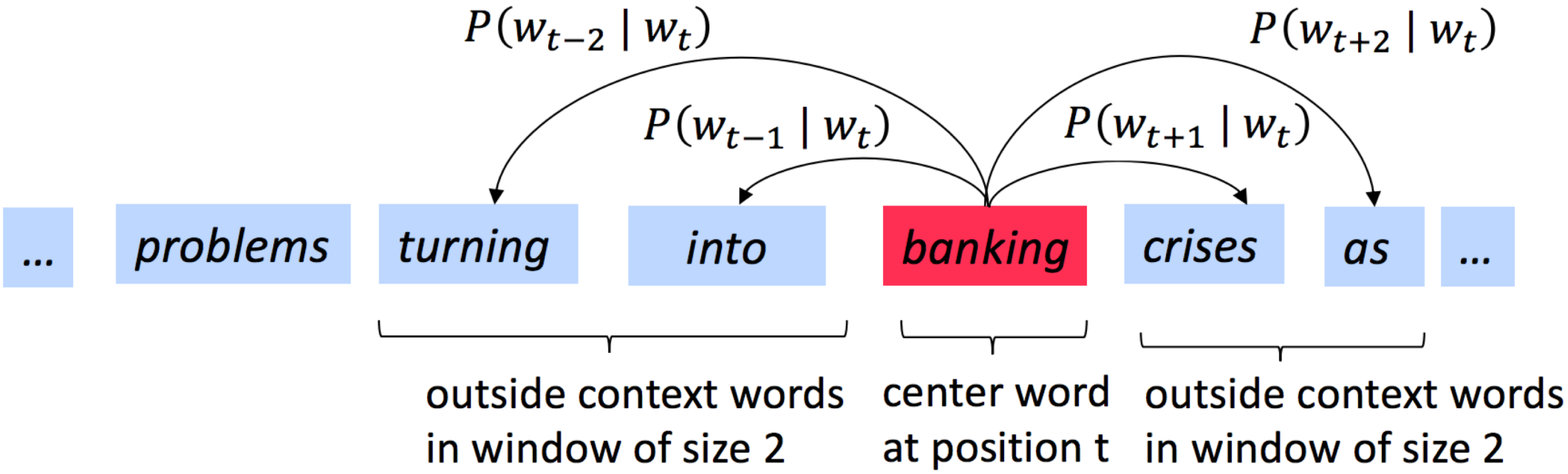
Task: Iterate through each word with a given window; for each word predict the context words within the window



(E.g. from Manning (2018) Stanford cs224n course)

Word2Vec (CBOW)

Task: Iterate through each word with a given window; for each word predict the context words within the window



(E.g. from Manning (2018) Stanford cs224n course)

Word2Vec (CBOW)

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.

Likelihood = $$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P\left(w_{t+j} \mid w_t; \theta\right)$$

θ is all variables to be optimized

sometimes called *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2Vec (CBOW)

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P\left(w_{t+j} \mid w_t; \theta\right)$$

θ is all variables to be optimized

sometimes called *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2Vec (CBOW)

We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Question: How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?

Answer: We will *use two* vectors per word *w*:

- v_w when *w* is a center word
- u_w when *w* is a context word

Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Word2Vec (CBOW)

We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Question: How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?

Answer: We will *use two* vectors per word *w*:

- v_w when *w* is a center word
- u_w when *w* is a context word

Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Word2Vec (CBOW)

Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product compares similarity of *o* and *c*.

$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

Normalize over entire vocabulary to give probability distribution

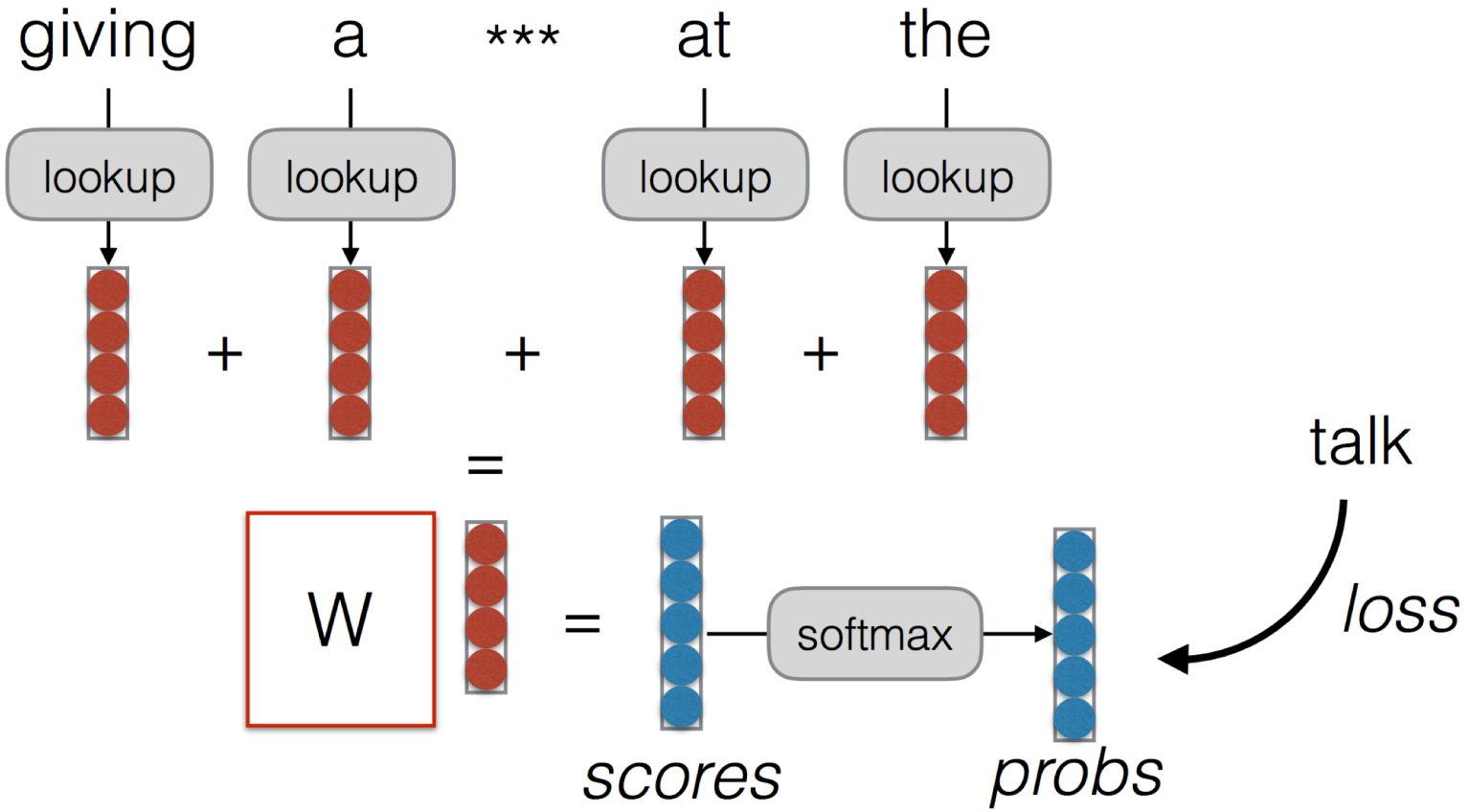
- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow \mathbb{R}^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

Word2Vec (CBOW)

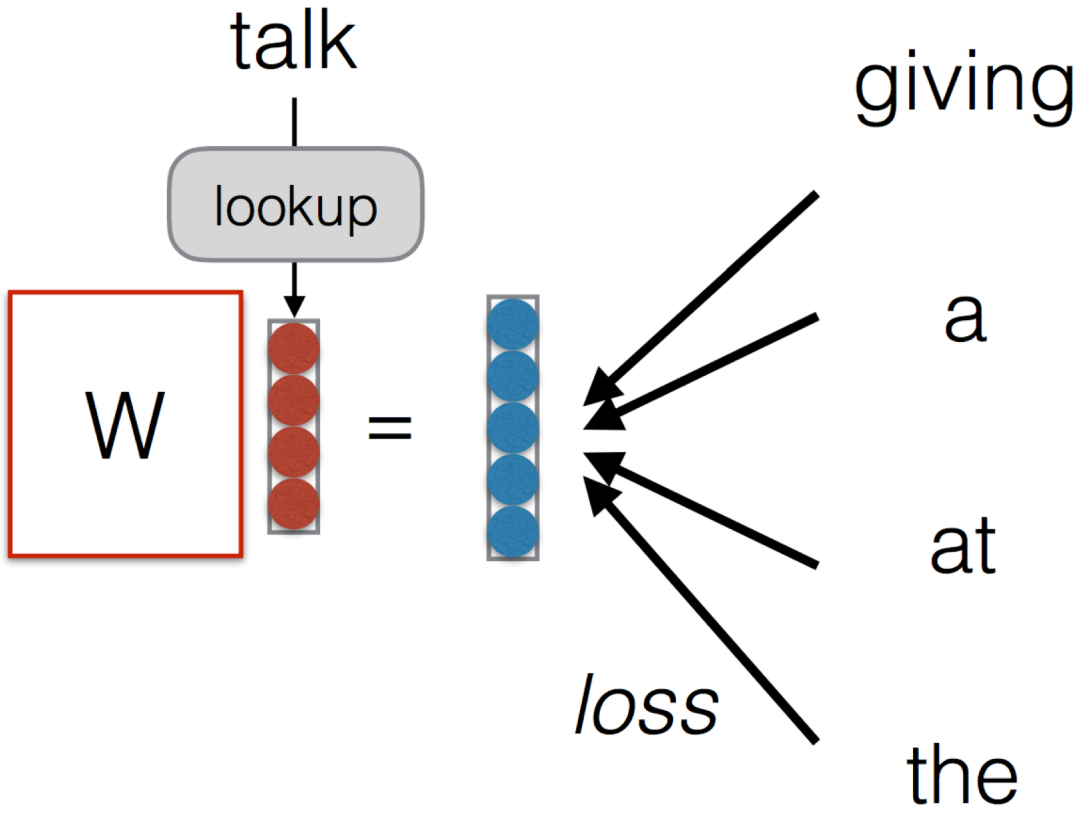
Task: Iterate through every word with a given window; learn W such the models can predict what's the word given only the context words as inputs.



(E.g. from Neubig (2018) CMU nn4nlp course)

Word2Vec (Skipgram)

Task: Iterate through every word with a given window; learn W such the models predicts 1.0 when the model is given (i) embeddings of the focus words and (ii) embedding of any word in the context and the models predicts 0.0 otherwise



(E.g. from Neubig (2018) CMU nn4nlp course)

Word2Vec (CBOW)

Sentence: the bulk of linguistic questions concern the dis- tinction between a and m. a linguistic account of phenomenon …

of	the bulk _____ linguistic questions
linguistic	bulk of _____ questions concern
questions	of linguistic _____ concern the
concern	linguistic questions _____ the dis-
the	questions concern _____ dis- tinction
dis-	concern the _____ tinction between
tinction	the dis- _____ between a
between	dis- tinction _____ a and
a	tinction between _____ and m.
and	between a _____ m. a
m.	a and _____ a linguistic
a	and m. _____ linguistic account
linguistic	m. a _____ account of
account	a linguistic _____ of a
of	linguistic account _____ a phenomenon
a	account of _____ phenomenon gen-
phenomenon	of a _____ gen- erally

Word2Vec (CBOW)

Sentence: the bulk of linguistic questions concern the dis- tinction between a and m. a linguistic account of phenomenon …

of	the bulk _____ linguistic questions
linguistic	bulk of _____ questions concern
questions	of linguistic _____ concern the
concern	linguistic questions _____ the dis-
the	questions concern _____ dis- tinction
dis-	concern the _____ tinction between
tinction	the dis- _____ between a
between	dis- tinction _____ a and
a	tinction between _____ and m.
and	between a _____ m. a
m.	a and _____ a linguistic
a	and m. _____ linguistic account
linguistic	m. a _____ account of
account	a linguistic _____ of a
of	linguistic account _____ a phenomenon
a	account of _____ phenomenon gen-
phenomenon	of a _____ gen- erally

Word2Vec (Skipgram)

Sentence: language users never choose words randomly , and language is essentially non-random .

In-/Outputs:

```
[(['language', 'users', 'choose', 'words'], 'never'),
(['users', 'never', 'words', 'randomly'], 'choose'),
(['never', 'choose', 'randomly', ','], 'words'),
(['choose', 'words', ',', 'and'], 'randomly'),
(['words', 'randomly', 'and', 'language'], ','),
(['randomly', ',', 'language', 'is'], 'and'),
([',', 'and', 'is', 'essentially'], 'language'),
(['and', 'language', 'essentially', 'non-random'], 'is'),
(['language', 'is', 'non-random', '.'], 'essentially')]
```

Word2Vec (Skipgram)

Sentence: `language users never choose words randomly , and language is essentially non-random .`

Windows:

```
['language', 'users', 'never', 'choose', 'words']

('never', 'language', 1),
('never', 'users', 1),
('never', 'choose', 1),
('never', 'words', 1),
('never', ',', 0),
('never', 'non-random', 0),
('never', 'is', 0),
('never', 'is', 0)
```

aka. negative sampling

Count vs Predict

- Strong connection between count-based and prediction based methods (Levy and Goldberg, 2014), so Skipgram ~= Matrix Factorization

- Skipgram shares similar global optimum as Matrix Factorization but != Matrix Factorization (Wilson, 2016)

- Skip-Gram – Zipf + Uniform = Vector Additivity (Gitten et al. 2017)

© 2018 National University of Singapore. All Rights Reserved

Count vs Predict

- Strong connection between count-based and prediction based methods (Levy and Goldberg, 2014), so Skipgram ~= Matrix Factorization

- Skipgram shares similar global optimum as Matrix Factorization but != Matrix Factorization (Wilson, 2016)

- Skip-Gram – Zipf + Uniform = Vector Additivity (Gitten et al. 2017)

© 2018 National University of Singapore. All Rights Reserved

How to Choose Context?

- **Different contexts lead to different embeddings**
- **Small context window:** more syntax related
- **Large context window:** more semantics related

© 2018 National University of Singapore. All Rights Reserved

How Good are my Embeddings?

© 2018 National University of Singapore. All Rights Reserved

Intrinsic Evaluation of Embeddings

- **Relatedness:** Measures correlations between embedding cosine similarity and human evaluation of similarity

- **Analogy:** "a is to b, as x is to ____"

- **Categorization:** Measure purity of clusters based on embeddings

- **Selectional Preference**: "tall" vs "high" man/building

© 2018 National University of Singapore. All Rights Reserved

Extrinsic Evaluation of Embeddings

- Load the pretrained embeddings
- Embed the input words as use the embeddings as input to models
- Evaluate which pre-trained embeddings are better for task X

© 2018 National University of Singapore. All Rights Reserved

When to use pre-trained embeddings?

- Generally, when you don't have much training/annotated data
- **Very Useful:** Use as inputs to model for classification task, e.g. tagging, parsing, textcat
- **Less Useful:** Machine Translation / Sequence generating tasks
- **Not Useful:** Generic Language Modeling, for those, we have sentence embedings…

© 2018 National University of Singapore. All Rights Reserved

How to make Embeddings Better?

© 2018 National University of Singapore. All Rights Reserved

Limitations

- **Sensitive to "tokens"** (cat vs cats)
- **Insensitive to polysemy** (Industrial *plant* vs "*I'm Groot*")
- **Inconsistent across space**, embeddings for the same words trained with different data are different
- **Can encode bias** (stereotypical gender roles, racial bias)
- **Not interpretable**

© 2018 National University of Singapore. All Rights Reserved

Embedding Bias

$$\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{king}} - \overrightarrow{\text{queen}}$$

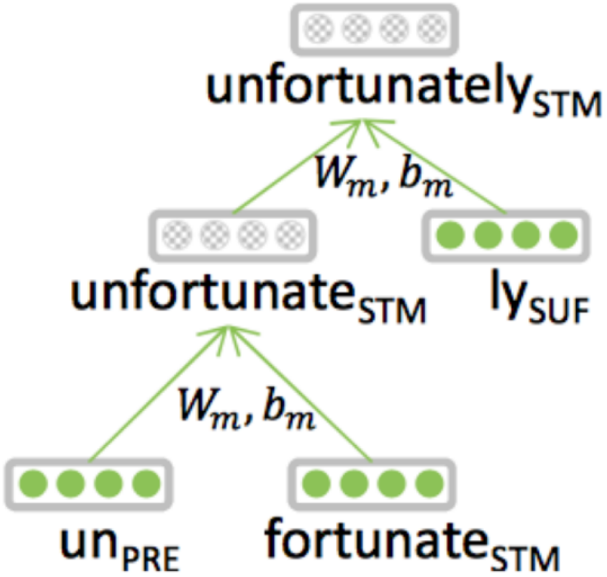
$$\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{computer programmer}} - \overrightarrow{\text{homemaker}}.$$

	"He" Occupations	"She" Occupations
Cosine Similarity	*["retired", "doctor", "teacher", "student", "miller", "assistant", "lawyer", "baker", "judge", "governor", "butler"]*	*["doctor", "**teacher**", "**nurse**", "actress", "student", "miller", "reporter", "retired", "lawyer", "actor", "artist"]*
Inner Product Similarity	*["cleric", "photographer", "**skipper**", "chaplain", "accountant", "inspector", "rector", "investigator", "psychologist", "treasurer", "supervisor"]*	*["**librarian**", "**housekeeper**", "**nanny**", "accountant", "sheriff", "envoy", "tutor", "salesman", "butler", "footballer", "solicitor"]*

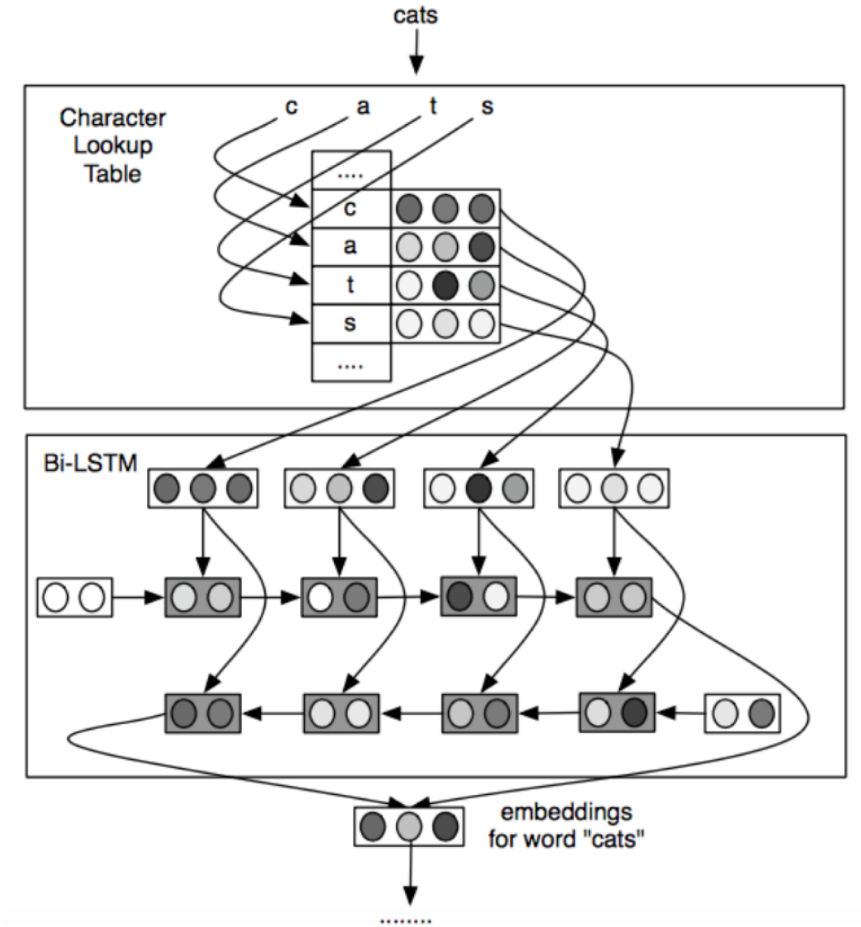
Non-Tokens Embeddings

- Can capture sub-word regularities

Morpheme-based
(Luong et al. 2013)



Character-based
(Ling et al. 2015)



© 2018 National University of Singapore. All Rights Reserved

Non-Tokens Embeddings

- **Bag of Characters Ngrams** (Bojanowski et al. 2016)

- `"where" -> ["<where>", "<wh", "her", "ere", "re>"]`

- Each word is the sum of all parts

- Embedding("where") = sum(Embedding(_ng) for _ng in ["<where>", "<wh", "her", "ere", "re>"])

© 2018 National University of Singapore. All Rights Reserved

De-biasing Embeddings

Extreme *she*	Extreme *he*
1. homemaker	1. maestro
2. nurse	2. skipper
3. receptionist	3. protege
4. librarian	4. philosopher
5. socialite	5. captain
6. hairdresser	6. architect
7. nanny	7. financier
8. bookkeeper	8. warrior
9. stylist	9. broadcaster
10. housekeeper	10. magician

Gender stereotype *she-he* analogies

sewing-carpentry	registered nurse-physician	housewife-shopkeeper
nurse-surgeon	interior designer-architect	softball-baseball
blond-burly	feminism-conservatism	cosmetics-pharmaceuticals
giggle-chuckle	vocalist-guitarist	petite-lanky
sassy-snappy	diva-superstar	charming-affable
volleyball-football	cupcakes-pizzas	lovely-brilliant

Gender appropriate *she-he* analogies

queen-king	sister-brother	mother-father
waitress-waiter	ovarian cancer-prostate cancer	convent-monastery

(Bolukbasi et al. 2016)

© 2018 National University of Singapore. All Rights Reserved

Word2Vec from Scratch

Sort of *"from scratch"*…

© 2018 National University of Singapore. All Rights Reserved

Environment Setup

Open Anaconda Navigator.

Go to the PyTorch installation page, copy the command as per configuration: https://pytorch.org/get-started/locally/

Fire up the terminal in Anaconda Navigator.

Start a Jupyter Notebook.

Download http://bit.ly/ANLP-Session3-Empty

Import the .ipynb to the Jupyter Notebook

© 2018 National University of Singapore. All Rights Reserved

3.0.1 Vocabulary

Given a text, the first thing to do is to build a vocabulary (i.e. a dictionary of unique words) and assign an index to each unique word.

```
import random
from itertools import chain

from tqdm import tqdm
from nltk import sent_tokenize, word_tokenize
from gensim.corpora import Dictionary

import torch
from torch import nn, optim, tensor, autograd
from torch.nn import functional as F
```

```

text = """Language users never choose words randomly, and language is essentially
non-random. Statistical hypothesis testing uses a null hypothesis, which
posits randomness. Hence, when we look at linguistic phenomena in corpora,
the null hypothesis will never be true. Moreover, where there is enough
data, we shall (almost) always be able to establish that it is not true. In
corpus studies, we frequently do have enough data, so the fact that a relation
between two phenomena is demonstrably non-random, does not support the inference
that it is not arbitrary. We present experimental evidence
of how arbitrary associations between word frequencies and corpora are
systematically non-random. We review literature in which hypothesis testing
has been used, and show how it has often led to unhelpful or misleading results.""".lower()

tokenized_text = [word_tokenize(sent) for sent in sent_tokenize(text)]

uniq_tokens = set(chain(*tokenized_text))

vocab = {}   # Assign indices to every word.
idx2tok = {} # Also keep an dict of index to words.
for i, token in enumerate(uniq_tokens):
    vocab[token] = i
    idx2tok[i] = token
```

```
text = """Language users never choose words randomly, and language is essentially
non-random. Statistical hypothesis testing uses a null hypothesis, which
posits randomness. Hence, when we look at linguistic phenomena in corpora,
the null hypothesis will never be true. Moreover, where there is enough
data, we shall (almost) always be able to establish that it is not true. In
corpus studies, we frequently do have enough data, so the fact that a relation
between two phenomena is demonstrably non-random, does not support the inference
that it is not arbitrary. We present experimental evidence
of how arbitrary associations between word frequencies and corpora are
systematically non-random. We review literature in which hypothesis testing
has been used, and show how it has often led to unhelpful or misleading results.""".lower()

tokenized_text = [word_tokenize(sent) for sent in sent_tokenize(text)]

uniq_tokens = set(chain(*tokenized_text))

vocab = {}   # Assign indices to every word.
idx2tok = {} # Also keep an dict of index to words.
for i, token in enumerate(uniq_tokens):
    vocab[token] = i
    idx2tok[i] = token
```

Using `gensim`, I would have written the above as such:

```
from gensim.corpora.dictionary import Dictionary
vocab = Dictionary(tokenized_text)
```

```
# Note the key-value order is different of gensim from the native Python's
dict(vocab.items())
```

```
 67: 'evidence',
 68: 'experimental',
 69: 'frequencies',
 70: 'how',
 71: 'of',
 72: 'present',
 73: 'systematically',
 74: 'word',
 75: 'been',
 76: 'has',
 77: 'led',
 78: 'literature',
 79: 'misleading',
 80: 'often',
 81: 'or',
 82: 'results',
 83: 'review',
 84: 'show',
 85: 'unhelpful',
 86: 'used'}
```

```
vocab.token2id['corpora']
```

```
23
```

```
vocab.doc2idx(sent0)
```

```
[6, 10, 7, 3, 11, 9, 0, 2, 6, 5, 4, 8, 1]
```

The "indexed form" of the tokens in the sentence forms the ***vectorized*** input to the `nn.Embedding` layer in PyTorch.

3.0.2 Dataset

Lets try creating a `torch.utils.data.Dataset` object.

```
from torch.utils.data import Dataset, DataLoader

class Text(Dataset):
    def __init__(self, tokenized_texts):
        """
        :param tokenized_texts: Tokenized text.
        :type tokenized_texts: list(list(str))
        """
        self.sents = tokenized_texts
        self.vocab = Dictionary(tokenized_text)

    def __getitem__(self, index):
        """
        The primary entry point for PyTorch datasets.
        This is were you access the specific data row you want.

        :param index: Index to the data point.
        :type index: int
        """
        return self.vectorize(self.sents[0])

    def vectorize(self, tokens):
        """
        :param tokens: Tokens that should be vectorized.
        :type tokens: list(str)
        """
        # See https://radimrehurek.com/gensim/corpora/dictionary.html#gensim.corpora.dictionary.Dictionary.doc2idx
        return {'x': self.vocab.doc2idx(tokens)}

    def unvectorize(self, indices):
        """
        :param indices: Converts the indices back to tokens.
        :type tokens: list(int)
        """
        return [self.vocab[i] for i in indices]
```

```
text_dataset = Text(tokenized_text)
```

```
text_dataset[0] # First sentence.
```

```
{'x': [6, 10, 7, 3, 11, 9, 0, 2, 6, 5, 4, 8, 1]}
```

```python
from torch.utils.data import Dataset, DataLoader

class Text(Dataset):
    def __init__(self, tokenized_texts):
        """
        :param tokenized_texts: Tokenized text.
        :type tokenized_texts: list(list(str))
        """
        self.sents = tokenized_texts
        self.vocab = Dictionary(tokenized_text)

    def __getitem__(self, index):
        """
        The primary entry point for PyTorch datasets.
        This is were you access the specific data row you want.

        :param index: Index to the data point.
        :type index: int
        """
        return self.vectorize(self.sents[0])

    def vectorize(self, tokens):
        """
        :param tokens: Tokens that should be vectorized.
        :type tokens: list(str)
        """
        # See https://radimrehurek.com/gensim/corpora/dictionary.html#gen
        return {'x': self.vocab.doc2idx(tokens)}

    def unvectorize(self, indices):
        """
        :param indices: Converts the indices back to tokens.
        :type tokens: list(int)
        """
        return [self.vocab[i] for i in indices]
```

Required function to access the data from Dataset object

```python
from torch.utils.data import Dataset, DataLoader

class LabeledText(Dataset):
    def __init__(self, tokenized_texts, labels):
        """
        :param tokenized_texts: Tokenized text.
        :type tokenized_texts: list(list(str))
        """
        self.sents = tokenized_texts
        self.labels = labels # Sentence level labels.
        self.vocab = Dictionary(self.sents)


    def __getitem__(self, index):
        """
        The primary entry point for PyTorch datasets.
        This is were you access the specific data row you want.

        :param index: Index to the data point.
        :type index: int
        """
        return {'X': self.vectorize(self.sents[index]), 'Y': self.labels[index]}

    def vectorize(self, tokens):
        """
        :param tokens: Tokens that should be vectorized.
        :type tokens: list(str)
        """
        # See https://radimrehurek.com/gensim/corpora/dictionary.html#gensim.corpora.
        return self.vocab.doc2idx(tokens)

    def unvectorize(self, indices):
        """
        :param indices: Converts the indices back to tokens.
        :type tokens: list(int)
        """
        return [self.vocab[i] for i in indices]
```

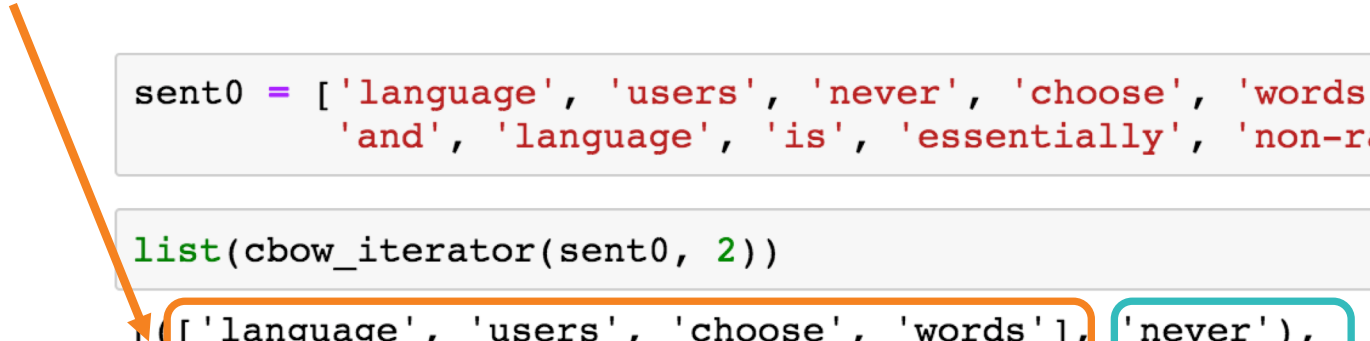
Overload the arguments and return the respective labels from the __getitem__()

3.1.1. CBOW

CBOW windows through the sentence and picks out the center word as the `Y` and the surrounding context words as the inputs `X`.

```
def per_window(sequence, n=1):
    """
    From http://stackoverflow.com/q/42220614/610569
        >>> list(per_window([1,2,3,4], n=2))
        [(1, 2), (2, 3), (3, 4)]
        >>> list(per_window([1,2,3,4], n=3))
        [(1, 2, 3), (2, 3, 4)]
    """
    start, stop = 0, n
    seq = list(sequence)
    while stop <= len(seq):
        yield seq[start:stop]
        start += 1
        stop += 1

def cbow_iterator(tokens, window_size):
    n = window_size * 2 + 1
    for window in per_window(tokens, n):
        target = window.pop(window_size)
        yield window, target    # X = window ; Y = target.
```

Context words



```
sent0 = ['language', 'users', 'never', 'choose', 'words', 'randomly', ',',
         'and', 'language', 'is', 'essentially', 'non-random', '.']
```

```
list(cbow_iterator(sent0, 2))
```

```
[(['language', 'users', 'choose', 'words'], 'never'),
 (['users', 'never', 'words', 'randomly'], 'choose'),
 (['never', 'choose', 'randomly', ',', 'and'], 'words'),
 (['choose', 'words', ',', 'and'], 'randomly'),
 (['words', 'randomly', 'and', 'language'], ','),
 (['randomly', ',', 'language', 'is'], 'and'),
 ([',', 'and', 'is', 'essentially'], 'language'),
 (['and', 'language', 'essentially', 'non-random'], 'is'),
 (['language', 'is', 'non-random', '.'], 'essentially')]
```

Focus words

```
list(cbow_iterator(sent0, 3))
```

```
[(['language', 'users', 'never', 'words', 'randomly', ','], 'choose'),
 (['users', 'never', 'choose', 'randomly', ',', 'and'], 'words'),
 (['never', 'choose', 'words', ',', 'and', 'language'], 'randomly'),
 (['choose', 'words', 'randomly', 'and', 'language', 'is'], ','),
 (['words', 'randomly', ',', 'language', 'is', 'essentially'], 'and'),
 (['randomly', ',', 'and', 'is', 'essentially', 'non-random'], 'language'),
 ([',', 'and', 'language', 'essentially', 'non-random', '.'], 'is')]
```

Summary

© 2018 National University of Singapore. All Rights Reserved

Embedding Checklist

- What is a Word2Vec model
- How to define CBOW and Skipgram task
- How to define CBOW and Skipgram models

© 2018 National University of Singapore. All Rights Reserved

PyTorch Checklist

- How to write a model and know what's going on behind loss.backward() and optimizer.step()
- How to declare your own torch.utils.data.Dataset object
- How to save/load a model
- How to overwrite weights and use pretrained embeddings
- Fine-tune/Unfreeze pre-trained embeddings

© 2018 National University of Singapore. All Rights Reserved

Fin

© 2018 National University of Singapore. All Rights Reserved